Electronic Notes in Theoretical Computer Science 6 (1997)
URL: http://www.elsevier.nl/locate/entcs/volume6.html 15 pages

Modularity in Denotational Semantics

John Power [1]

Laboratory for the Foundations of Computer Science
University of Edinburgh
Edinburgh, Scotland

Abstract

We consider a modular approach to denotational semantics. We reformulate and extend the idea of monads as notions of computation to algebraic structure together with a construction of an extended semantic category. We show that upon making that reformulation, one can obtain some account of modularity, in particular accounting for the interaction between side-effects and various forms of nondeterminism. That involves extending the notion of distributivity of a monad over another monad to distributivity of a monad over any algebraic structure. We give a general theorem which asserts when algebraic structure extends along a Kleisli category, thus allowing modularity.

1 Introduction

Some years ago, Eugenio Moggi proposed the use of monads to model what he called notions of computation [9]. The idea is that to each of many possible features of programming languages, such as nondeterminism, side-effects, exceptions, or continuations, one would ascribe a monad on a given base category A. For instance, A would typically be taken to be the category of ω-cpo's. A monad for nondeterminism would be given by any of the three classical powerdomains. A monad for side-effects would be given by $S \to (S \times -)$, where S was an object of A that represented an object of states. A monad for exceptions was given by $- + E$ for an object E of exceptions. And a monad for continuations was given by $(- \to Ans) \to Ans$, where Ans was a type of possible answers, or results of computation.

The central goal of that work was to provide an account of modularity in programming. One sought a general construction that, to each pair of monads, possibly subject to some coherence, would yield another monad; and this general construction should, upon taking a powerdomain P and the side-effects

[1] This work is supported by EPSRC grant GR/J84205: Frameworks for programming language semantics and logic.

monad $S \to (S \times -)$, yield a reasonable semantics for nondeterminism together with side-effects. A reasonable semantics was deemed to be that given by the monad $S \to P(S \times -)$: a program would input a pair consisting of a state s and a value x, and would yield a state s' together with a value x', but the nondeterminism led one to introduce the powerdomain to represent the possible pairs that may emerge. However, the monad $S \to P(S \times -)$ is not given by a composite of the two monads in either order; nor is it given by their coproduct; and there seems no known general construction on a pair of monads that accounts for it. Moreover, the same is true for side-effects combined with several of the other monads, for instance that for probabalistic nondeterminism. It also applies to continuations together with nondeterminism.

Moggi recognized that difficulty with monads, so introduced the notion of monad transformer, a monad transformer on a category A being an endofunction on the set of monads on A (see Cenciarelli's thesis [2], Cenciarelli and Moggi's [3] and Moggi's [10]). For instance, if one wants to add side-effects to a monad, one might consider the monad transformer that takes a monad T to the composite $S \to T(S \times -)$. So the side-effects monad is no longer taken as primitive. More generally, implicit in the definition of monad transformer is the idea that a monad should not be taken as primitive, as it is derivable from a monad transformer $\mathcal{F}$ by applying $\mathcal{F}$ to the identity monad. That leaves the question: what more primitive structure than a monad should be used to model notions of computation in a modular fashion? An answer must implicitly address the question of finding a general construction of monad transformers. In fact, our formulation is more subtle, so does not explicitly yield monad transformers, primarily for two reasons. First, monad transformers are global, applying to all monads, whereas one would expect a coherence requirement before applying a transformer. For instance, given a monad T', the operation taking a monad T to the composite TT' should in spirit be a monad transformer, which one might use if T' was the monad $- + E$ for exceptions. However, it is not, as one does not always have a distributive law. Second, since we must replace a monad by some more primitive structure anyway, it is more natural to apply a transformer to the more primitive structure directly rather than to a monad induced by the structure. So, for these two reasons, and also because our work is formulated a little differently anyway, the specific notion of monad transformer does not arise explicitly here.

In this paper, we start to give an account of modularity based upon a restructuring of the semantics of notions of computation, subsuming, but strictly more general than, Moggi's monadic approach [9]. The idea is that, instead of regarding a monad as modelling a notion of computation, we consider a notion of computation to be modelled by algebraic structure on a category A, together with a construction using that algebraic structure, of a pair (B, j) consisting of a category B and an identity on objects functor $j : A \longrightarrow B$. For instance, for side-effects, the algebraic structure (see below for more detail) may be that for a small symmetric monoidal category together with a

specified object S; then, using that structure, one constructs B by putting $B(a,b) = A(S \otimes a, S \otimes b)$, and defining composition by that in A, with the functor $j : A \longrightarrow B$ given on homs by $S \otimes -$. This approach is in line with the reformulation of Moggi's structures by the author and Edmund Robinson in [16] (see also [15]). In no way do we complete an analysis of this proposal here, but we make a start that does account for our leading example of modularity above and does extend the construction given by a composite of monads.

More precisely, we base our study upon finitary 2-monads M on Cat and their algebras. These are equivalent, in a precise sense, to algebraic structure on Cat and universal algebras. General treatments of this, directed towards computer scientists, appear in [17] and [14]. Here, we include an appendix to recall the details as they apply to us. For instance, there is a 2-monad M for which an M-algebra (A, a) is precisely a small category A with finite products. There is another 2-monad for which an algebra is precisely a small monoidal category. There is another for which an algebra is precisely a small category with a monad on it. There is another for which it consists of a small symmetric monoidal category together with an object S; and another for which it is a small category with finite coproducts together with an object E. With one exception, such 2-monads allow us to construct Kleisli categories for all of the monads we have mentioned above, starting from their elementary definitions. The sole exception is that for continuations: it involves contravariance, which we shall not address here. But observe, even there, if we are given the monad $(- \to Ans) \to Ans$ as the given primitive algebraic structure, we do incorporate the construction of the Kleisli category as an example.

Given a 2-monad M and an M-algebra (A, a), we consider, in Section 2, a monad T on A, and we give a condition under which the M-structure lifts from A to the Kleisli category $Kl(T)$. The condition is necessarily nontrivial. For instance, there is a 2-monad for which an algebra is a small category with binary products (see Example A.2), but binary products typically do not lift along the Kleisli construction. But what lifting does imply, when true, is that any construction on A that only uses the given M-structure lifts to a construction on $Kl(T)$. For instance, consider the 2-monad for a small category with finite coproducts together with an object E. That structure, on any small category A, lifts to $Kl(T)$ for any monad T. So the construction $Kl(- + j(E))$ may be made on the category $Kl(T)$, extending that given on A. So if, for example, T was given by a powerdomain P, then one could consider the category constructed by first moving to $Kl(P)$, then applying the construction $Kl(- + j(E))$ to it. This gives exactly the category one seeks for modelling nondeterminism together with exceptions. One can study side-effects together with nondeterminism similarly: in order to make the construction of the semantic category for side-effects, it suffices to have symmetric monoidal structure on A together with an object S; one does not need an exponential to construct the semantic category (see [16]).

One thus obtains the category we sought. The condition cited here, and the proof that it does the job, is the main result of the paper. The two leading examples for us are that for side-effects and nondeterminism, and in general, the composite of two monads, for instance those for nondeterminism and probabalistic nondeterminism.

There are several points to note here. The most important is that this paper does not give anything near a full solution to the problem of modularity, but rather makes a first step. The results and claims are limited when compared to a grand aim of fully achieving modularity in denotational descriptions. In particular, a modular approach is unlikely to yield penetrating insight into subtle computational phenomena, especially when such arise from combining two or more features that are not orthogonal. What can be hoped for is a combination of two notions of computation that yields a model that is comparable to that which would be obtained if a human were to try to construct a naive model. For instance, although side-effects are our leading example, we do not claim to address the central issues of modelling state: those of local variables and lack of snapback (see O'Hearn and Tennent's [11]). A more technical point is that our analysis is clearly incomplete in that, to model modularity, we take algebraic structure together with a construction to model one notion of computation, whereas we take a monad to model the other. This is also the reason why it is not immediately possible to iterate our construction. Obviously, the more natural and more general situation would be to consider two algebraic structures and two constructions. On the other hand, one can typically take a $j : A \longrightarrow B$ as we have here, and embed it into some $j' : A' \longrightarrow B'$ that has a right adjoint, hence gives a monad, while respecting structure (see [16,15] for some examples). But also, we have not specified what sort of constructions we are allowed to make here. All of that analysis, together with that for contravariance, we leave for future work. Finally, note that this analysis is not commutative: we lifted the construction for side-effects along $Kl(P)$, not the other way around. It is routine to check that it makes a difference.

We do not address one important point here at all. In [16], it was shown that strong monads give rise to premonoidal categories. We believe that premonoidal categories are a fundamental structure in this regard for modelling contexts. However, premonoidal structure is not algebraic structure on Cat in the precise sense we use here. There are several ways to deal with that (for instance, see [15]), but we defer it because we already introduce several complex new notions, and also because our current ideas on its relationship with modularity do not yet fit into this framework as well as we should like.

I am unaware of any other work in category theoretic approaches to denotational semantics that addresses modularity, except for that of Moggi and Cenciarelli. The term modularity is used in algebraic specification too, but we do not address that at all. Also, monad transformers have been used in interpreters and compiler construction, for instance in [8] and [18], but that is

not semantic, which is our concern here.

The paper is organized as follows. In Section 2, we analyse the category of algebras and lax maps of algebras for a 2-monad on Cat, leading to our definition of a distributive law, and some analysis of it. Then in Section 3, we show how a distributive law allows us to extend algebraic structure from a base category to the Kleisli category of a monad on it.

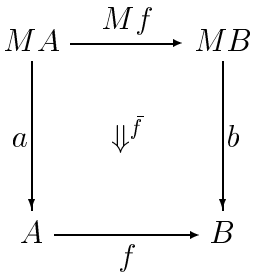
2 Distributivity

There has long been a notion of a distributive law between two monads. Given monads T and T', it is exactly that extra data and axioms required to make the composite functor TT' into a monad, respecting the structure of its component monads. But a small category with a monad is an instance of algebraic structure, i.e., it is an algebra for a 2-monad M on Cat. So we could consider generalising the notion of distributive law to consider distributing a monad over any algebraic structure. That is precisely what we do in this section. In order to do that, it is mathematically elegant to give a little development of 2-categories of algebras for a 2-monad on a 2-category, in particular Cat. For more detail, one should see [1] for the mathematics and [14] for an overview of some of the ideas, directed towards computer scientists.

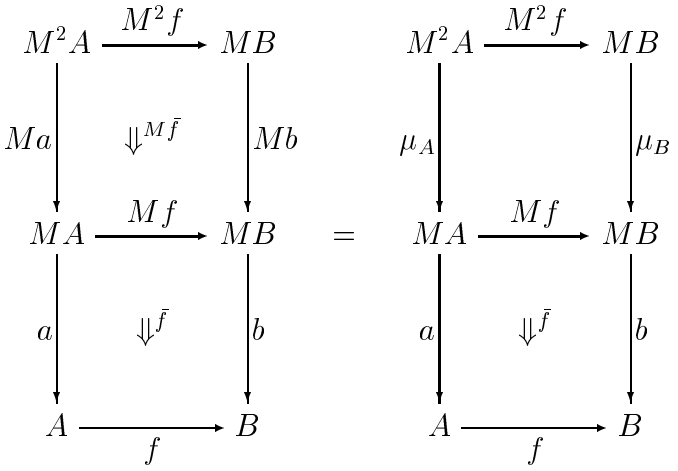
Given any 2-monad M on Cat, or more generally on any 2-category C, one can define the 2-category of algebras and what are usually called *lax* maps of algebras. In order to understand the notation used below, consider the left diagram below that contains M^2A. It uses a notation that has proved particularly apt when one needs 2-categories, as we do here. The diagram represents a composite 2-cell: the domain is given by the composite of the three 1-cells from the top left, along the top, then down to the bottom right; the codomain is given by the composite of the three 1-cells from the top left, down to the bottom left, then along to the bottom right. The composite 2-cell is given by first composing $M\bar{f}$ with b (which is just a composite of a natural transformation with a functor), doing dually with $\bar{f}$ and Ma, then taking the *vertical* composite, i.e., the pointwise composite of the two natural transformations we have just described. The other diagrams should be read similarly. This diagramatic presentation sometimes goes under the name of *pasting* composition (see [7,13]). Any square containing no label inside it is to be read as the identity 2-cell. It follows from the equations for algebras that we are only asserting the equality of natural transformations for which we already know their domains are equal, and their codomains are equal. This notation is used freely in the literature on 2-categories and can be found in most texts that use them, e.g. [1].

Definition 2.1 *Given M-algebras (A, a) and (B, b), a* lax *map of algebras from (A, a) to (B, b) consists of a 1-cell $f : A \longrightarrow B$ in C (i.e., a functor),*

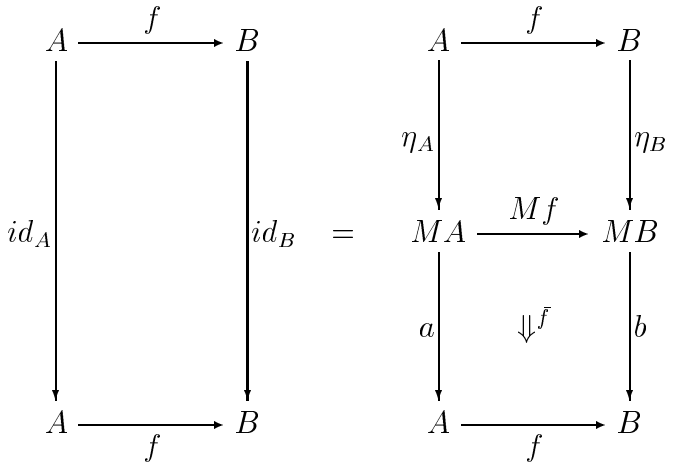
together with a 2-cell (i.e., a natural transformation)



such that



and



A 2-cell *between lax maps* $(f, \bar{f})$ *and* $(g, \bar{g})$ *is a 2-cell in* C *between* f *and* g *that respects* $\bar{f}$ *and* $\bar{g}$.

By using the composition of C, it follows that M-algebras, lax maps of M-algebras, and 2-cells form a 2-category, which we shall denote by $M - Alg_l$. This agrees with the notation in the relevant literature, and this situation has undergone extensive study, in particular in [1].

That is all that is needed, together with the examples as in the introduction and below, in order to understand this paper. However, if we wish to explain lax maps directly in terms of algebraic structure, we may do so based upon an understanding of the appendix here and its notation. Given algebraic structure (S, E) on C, we know from Appendix A that there is a 2-monad M on C for which $(S, E) - Alg$ is equivalent to the category of algebras for the 2-monad M. In fact, we can go further and characterize the lax maps of algebras in terms of operations and equations too [6]. A lax map of algebras $(f, \phi) : (A, \nu) \longrightarrow (B, \delta)$ amounts to a 1-cell $f : A \longrightarrow B$ together with a natural transformation

$$\begin{array}{ccc} C(c,A) & \xrightarrow{C(c,f)} & C(c,B) \\ {\scriptstyle \nu_c}\downarrow & \Downarrow^{\phi_c} & \downarrow{\scriptstyle \delta_c} \\ C(Sc,A) & \xrightarrow[C(Sc,f)]{} & C(Sc,B) \end{array}$$

subject to a condition to the effect that the ϕ_c's respect the equations E. To see that condition explicitly, simply follow the definition of (S, E)-algebra as in Appendix A, see precisely where it uses the equations, and insert the equations in the corresponding places here. This is done in detail (except that the ϕ_c's are assumed to be isomorphisms) in [6].

A 2-cell between lax maps (f, ϕ) and (g, ψ) amounts to a 2-cell $\alpha : f \Rightarrow g$ that respects ϕ and ψ.

Definition 2.2 *Given a 2-monad M on C, we call a monad in the 2-category $M - Alg_l$ a* monad with a distributive law over M.

More concretely, a monad with a distributive law over M amounts to an M-algebra (A, a) together with a monad (T, m, j) on A and a 2-cell

$$\begin{array}{ccc} MA & \xrightarrow{MT} & MA \\ {\scriptstyle a}\downarrow & \Downarrow^{t} & \downarrow{\scriptstyle a} \\ A & \xrightarrow[T]{} & A \end{array}$$

such that the two diagrams making (T, t) a lax map commute, as do the two diagrams making m and j 2-cells in $M - Alg_l$.

It follows from the above analysis that given algebraic structure (S, E) on C, we can express all this data and these axioms directly in terms of the algebraic structure.

Example 2.3 *Consider the 2-monad M on Cat for which an algebra consists of a small category A with a monad T' on it. Then, a monad with a distributive law over M amounts to a monad T with a distributive law of T' over T in the usual sense.*

For another example

Example 2.4 *Consider the 2-monad M for which an algebra is a small symmetric monoidal category. Then to give a monad T with a commutative strength is equivalent to giving a monad with a distributive law over M.*

As mentioned in the introduction, we also have

Example 2.5 *Let M be the 2-monad for which an algebra is a small category with finite coproducts. Then every monad T on an the underlying category of an M-algebra has a unique distributive law over M, determined by finite coproducts.*

3 Extending algebraic structure to a Kleisli category

A distributive law, as we have just introduced, is the structure we seek for an account of modularity. However, we need an additional axiom in order to allow the lifting we desire of algebraic structure from a base category A to the Kleisli category $Kl(T)$. Again, we first need a little more analysis of the 2-category of algebras for a 2-monad M on Cat, or more generally, on a 2-category with a little cocompleteness. We shall continue to restrict our attention to Cat.

Say a 2-monad M *preserves Kleisli objects* if given a monad T in C, if one takes its Kleisli category, then applies M, one obtains the same result, up to coherent isomorphism, as first applying M to the monad, then taking the Kleisli category of MT. We have

Proposition 3.1 *If M preserves Kleisli objects, then for any M-algebra (A, a) and any monad T on A with a distributive law over M, the category $Kl(T)$ has an M-algebra structure such that the canonical functor $j : A \longrightarrow Kl(T)$ preserves the M-structure.*

Proof. This is routine. It follows from the universal property of the Kleisli category and in fact extends to any class of conical lax colimits. □

In fact (see [4]), any left adjoint in a category of the form $M - Alg_l$ always preserves the M-structure up to coherent isomorphism. Moreover, there is a converse: given an adjunction between the underlying ordinary categories of two M-algebras, the adjunction lifts to $M - Alg_l$ if and only if the left adjoint preserves M-structure. That is resaonably straightforward, but one must take care with the coherence. It follows that there is a converse to Proposition 3.1. This is a routine deduction, but it is central for us.

Theorem 3.2 *Supppose M preserves Kleisli objects. Then, given an M-algebra (A, a) and a monad T on A, there is a bijection between M-structures on $Kl(T)$ that respect j and distributive laws of T over M.*

Finally, the prospect of starting with algebraic structure (S, E), finding a description of the 2-monad it generates, then checking whether it preserves Kleisli objects, is unappealing. So we seek a condition which implies that, together with some examples in which the condition holds. Since M is a 2-monad, it must preserve adjunctions. So, a sufficient condition for M to preserve Kleisli objects is the condition that M preserves bijective on objects functors. Every example of algebraic structure we have considered in the paper does that. All follow from fairly routine calculation: one must give a description of the free M-algebra on a small category. So for instance, the free category with (strictly associative) finite products on an ordinary category X has as an object, a list of objects of X. That is independent of the arrows in X; so the 2-monad with such algebras preserves bijective on objects functors. Similar proofs apply to monoidal categories, symmetric monoidal categories, and categories with finite coproducts. The proof for a small category with an object (such as S or E) is obviously almost trivial, and putting that together with the above structures is routine. For a small cateory with a monad on it, it is again straightforward: one freely generates the objects by freely applying a monad T to each object of X, and continuing inductively; and again that is independent of the arrows of X.

We finally return to our leading example.

Example 3.3 *Let M be the 2-monad for which an M-algebra is a small symmetric monoidal category A together with a specified object S. As mentioned above, M preserves bijective-on-objects functors, and therefore preserves Kleisli objects. Let P be the monad given by any of the classical powerdomains. Then P is commutative, and so by Example 2.4 there is a distributive law of P over M. Thus by Theorem 3.2, also by Proposition 3.1, $Kl(P)$ has an M-algebra structure, i.e., $Kl(P)$ is a small symmetric monoidal category with a specified object, with M-structure preserved by $j : A \longrightarrow Kl(P)$. So the specified object of $Kl(P)$ is $j(S)$. Since $Kl(P)$ has M-structure, one can apply the side-effects construction to it, yielding a category $Kl(P)_S$ whose objects are those of A, with a map from a to b given by a map in $Kl(P)$ from $j(S) \otimes a$ to $j(S) \otimes b$, i.e., a map in A from $S \otimes a$ to $P(S \otimes b)$, which was as we desire.*

For another example,

Example 3.4 *Let M be the 2-monad for which an M-algebra is a small category A with finite coproducts and a specified object E. As mentioned above, M preserves bijective-on-objects functors, hence preserves Kleisli objects. Let P be a powerdomain. By Example 2.5, it follows that P, as any monad, has a distributive law over M. So by Theorem 3.2 or by Proposition 3.1, $Kl(P)$*

has finite coproducts and a specified object given by $j(E)$. So one can apply the exceptions construction to it, yielding a category $Kl(P)_E$ whose objects are those of A, and in which a map from a to b is a map in $Kl(P)$ from a to $b + E$, i.e., a map in A from a to $P(b + E)$, which is as we desire.

For a final example,

Example 3.5 *Let M be the 2-monad for which an M-algebra is a small category A together with a monad T' on it. As mentioned above, M preserves bijective-on-objects functors, and therefore preserves Kleisli objects. Let T be any monad, and suppose there is a distributive law of T' (for a given M-algebra) over T. Then by Example 2.3, there is a distributive law of T over M. So by Theorem 3.2 or Proposition 3.1, $Kl(T)$ has M-structure with $j : A \longrightarrow Kl(T)$ preserving M-structure. So we can apply the Kleisli construction for $Kl(T')$ to $Kl(T)$, yielding a category $Kl(T)_{T'}$ whose objects are those of A and in which an arrow from a to b is an arrow in $Kl(T)$ from a to $T'(b)$, i.e., an arrow in A from a to $TT'(b)$ as desired.*

References

[1] R. Blackwell, G.M. Kelly, and A.J. Power, Two-dimensional monad theory, J. Pure Appl. Algebra 59 (1989) 1–41.

[2] Pietro Cenciarelli, Computational applications of calculi based on monads, Edinburgh Ph.D. thesis, CST–127–96 (1996).

[3] Pietro Cenciarelli and Eugenio Moggi, A syntactic approach to modularity in denotational semantics, Proc CTCS 93.

[4] G.M. Kelly, Doctrinal adjunction, Sydney Category Seminar, Lecture Notes in Mathematics 420 (1974) 257–280.

[5] G.M. Kelly and A.J. Power, Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads, J. Pure Appl. Algebra 89 (1993) 163–179.

[6] G.M. Kelly and A.J. Power, Algebraic structure in the enriched context, (1988) (draft).

[7] G.M. Kelly and R. Street, Review of the elements of 2-categories, Sydney Category Seminar, Lecture Notes in Mathematics 420 (1974) 75–103.

[8] S. Liang, M. Jones, and P. Hudak, Monad transformers and modular interpreters, Proc POPL 95, ACM Press, San Francisco (1995) 333–343.

[9] E. Moggi, Notions of computation and monads, Information and Computation 93 (1991) 55–92.

[10] E. Moggi, Metalanguages and applications, in Semantics and Logic of Computation, A.M. Pitts, P. Dyjber (eds.), Publications of the Newton Institute, CUP (1997).

[11] P. O'Hearn and R.D. Tennent, Parametricity and local variables, J. ACM 42 (1995) 658–709.

[12] A.J. Power, An algebraic formulation for data refinement, Proc MFPS '89, Lecture Notes in Computer Science 442 (1989) 390–401.

[13] A.J. Power, A 2-categorical pasting theorem, J. Pure Appl. Algebra 129 (1990) 439–445.

[14] A.J. Power, Why tricategories? Information and Computation 120 (1995) 251–262.

[15] A.J. Power, Premonoidal categories as categories with algebraic structure (submitted).

[16] A.J. Power and E.P. Robinson, Premonoidal categories and notions of computation, Proc. LDPL '96, Math Structures in Computer Science (to appear).

[17] Edmund Robinson, Variations on algebra: monadicity and generalisations of algebraic theories, Math. Structures in Computer Science (to appear).

[18] Guy L. Steele, Building interpreters by composing monads, POPL 94, ACM Press, San Francisco (1994).

A Algebraic Structure

In this appendix, we define *algebraic structure* in the 2-category Cat. One can define algebraic structure in any locally finitely presentable 2-category, but we would need to introduce the concept of tensor in a 2-category to do so. In the case of primary interest to us, Cat, we can avoid introducing tensors. So we avoid them. In what follows, we let C denote Cat. Also, we refer to finitely presentable objects. In order to understand this paper, one does not need a precise understanding of finitely presentable objects: all one needs to know is that they include all finite categories. The only reason we mention finitely presentable objects here is in order to give an accurate statement of the converse of the theorem; but we do not use the converse. For a gentler account of algebraic structure, see Robinson's paper [17]. See also [15] for further analysis of the use that can be made of algebraic structure in Cat.

Algebraic structure in Cat extends the familiar notions of algebraic structure with respect to Set. If one replaced 2-categories by ordinary categories, 2-functors by functors, 2-natural transformations by natural transformations, categories by sets, and functors by functions, in all that followed, one would have precisely the familiar account of algebraic structure in Set, with the algebras being precisely universal algebras. So for instance, $ob\,C_f$, which we are about to define, would become the set of all finite sets; the category $S(c)$ would become a set; and the construction $F(S)$ would become the usual construction of all derived operations given by basic operations S.

Let $ob\,C_f$ denote the discrete 2-category on the set of (isomorphism classes of) finitely presentable objects in C. Then a *signature* on C consist of a 2-functor S: $ob\,C_f \longrightarrow C$. For each $c \in ob\,C_f$, $S(c)$ is called the object of *basic operations of arity* c.

From S, we construct $S_\omega : C_f \longrightarrow C$ as follows: set

$$S_0 = J, \quad \text{the inclusion of } C_f \text{ in } C$$
$$S_{n+1} = J + \sum_{d \in ob\,C_f} C(d, S_n(-)) \times S(d),$$

and define

$$\sigma_0 : S_0 \longrightarrow S_1 \quad \text{to be} \quad inj : J \longrightarrow J + \textstyle\sum_{d \in ob\,C_f} C(d, S_0(-)) \times S(d)$$
$$\sigma_n = J + \textstyle\sum_{d \in ob\,C_f} C(d, \sigma_{n-1}(-)) \times S(d) : S_n \longrightarrow S_{n+1}$$
$$S_\omega = colim_{n<\omega} S_n.$$

In many cases of interest, each σ_n is a monomorphism, so S_ω is the union of $\{\, S_n \,\}_{n<\omega}$. For each c, we call $S_\omega(c)$ the object of *derived c-ary operations.*

A signature is typically accompanied by equations between derived operations. So we define the *equations* of an algebraic theory with signature S to consist of a 2-functor $E : ob\,C_f \longrightarrow C$ together with 2-natural transformations $\tau_1, \tau_2 : E \longrightarrow S_\omega(K(-))$, where $K : ob\,C_f \longrightarrow C_f$ is the inclusion.

Algebraic structure on C consists of a signature S, together with equations (E, τ_1, τ_2). We generally denote algebraic structure by (S, E), suppressing τ_1 and τ_2.

We now define the algebras for a given algebraic structure. Given a signature S, an *S-algebra* consists of an object A of C together with a functor $\nu_c : C(c, A) \longrightarrow C(S(c), A)$ for each c. So, an S-algebra consists of a carrier A and an interpretation of the basic operations of the signature. This interpretation extends canonically to the derived operations, giving an $S_\omega(K(-))$-algebra, as follows:

$$\nu_0 : C(c, A) \longrightarrow C(S_0(c), A) \quad \text{is the identity;}$$

to give $\nu_{n+1} : C(c, A) \longrightarrow C(S_{n+1}(c), A)$, using the fact that $C(-, A)$ preserves colimits, is to give a functor $C(c, A) \longrightarrow C(c, A)$, which we will make the identity, and for each d in $ob\,C_f$, a functor

$$C(c, A) \longrightarrow [C(d, S_n(c)), C(S(d), A)],$$

or equivalently, $C(c, A) \times C(d, S_n(c)) \longrightarrow C(S(d), A)$, which is given inductively by

$$C(c, A) \times C(d, S_n(c)) \xrightarrow{\nu_n \times id} C(S_n(c), A) \times C(d, S_n(c)) \xrightarrow{comp} C(d, A) \xrightarrow{\nu} C(S(d), A)$$

Given signature S and equations E, an *(S, E)-algebra* is an S-algebra that satisfies the equations, i.e., an S-algebra (A, ν) such that both legs of

$$C(c, A) \xrightarrow{\nu_\omega} C(S_\omega(Kc), A) \underset{C(\tau_{2c}, A)}{\overset{C(\tau_{1c}, A)}{\rightrightarrows}} C(E(c), A)$$

agree.

Given (S,E)-algebras (A,ν) and (B,δ), we define the homcategory $(S,E)-Alg((A,\nu),(B,\delta))$ to be the equaliser in Cat of

$$\begin{array}{ccc} C(A,B) & \xrightarrow{\{C(c,-)\}_{c\in obC_f}} & \prod_c [C(c,A),C(c,B)] \\ \downarrow{\scriptstyle \{C(S(c),-)\}_{c\in obC_f}} & & \downarrow{\scriptstyle \prod_c[C(c,A),\delta_c]} \\ \prod_c [C(S(c),A),C(S(c),B)] & \xrightarrow[\prod_c[\nu_c,C(S(c),B)]]{} & \prod_c [C(c,A),C(S(c),B)]. \end{array} \tag{A.1}$$

This agrees with our usual universal algebraic understanding of the notion of homomorphism of algebras, internalising it to Cat. $(S,E)-Alg$ can then be made into a 2-category in which composition is induced by that in C. An arrow in $(S,E)-Alg$ is a functor $f : A \longrightarrow B$ such that for all finitely presentable c, $f\nu_c(-) = \delta_c(f-) : C(c,A) \longrightarrow C(S(c),B)$, i.e., a functor that commutes with all basic c-ary operations for all c.

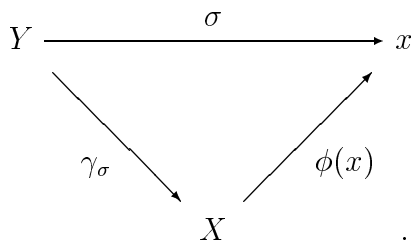
The special case of the main result of [5] that is central to our work is

Theorem A.1 *A 2-category is equivalent to $(S,E)-Alg$ for algebraic structure (S,E) on C if and only if there is a finitary 2-monad M on C such that the 2-category is equivalent to $M-Alg$.*

For an example of algebraic structure, see how the category of small categories with binary products is given by algebraic structure.

Example A.2 *Let 2 denote the discrete category on two objects; let $\rightarrow$ denote the arrow category; let Cone denote the category given by two objects together with a cone over them; and let Doublecone denote Cone together with a cone over it, thus a pair of objects, a pair of cones over them, and an intermediary map from one vertex to the other, commuting with the projections. Now define $S : obC_f \longrightarrow C$ by $S(2) = Cone$, $S(Cone) = Doublecone$, and for all other c, $S(c)$ is the empty category.*

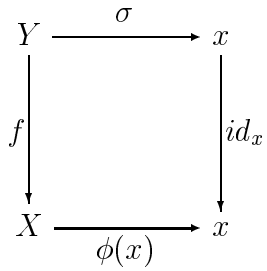
An S-algebra is a small category A together with a functor $\phi : [2,A] \longrightarrow [Cone,A]$ and a functor $\psi : [Cone,A] \longrightarrow [Doublecone,A]$. The functor ϕ is to take a pair of objects to its limiting cone, and the functor ψ is to take a cone to itself, the limiting cone, and the unique comparison map. So we add equations as follows: we may add equations factoring through $S_1(2)$ and $S_1(Cone)$ respectively so that $\phi(x) : Cone \longrightarrow A$ restricts along the inclusion $2 \longrightarrow Cone$ to x, and so that ψ sends a cone $\sigma : Y \longrightarrow x$ to a commutative diagram of the form



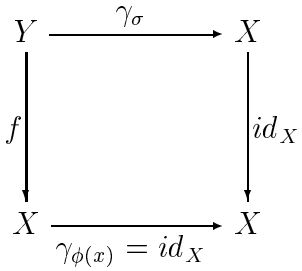
Finally, we add an equation factoring through $S_2(2)$ so that, for each $x : 2 \longrightarrow A$, we have $\gamma_{\phi(x)} = id_X$.

Putting this together, we put $E(2) = Cone+ \rightarrow$, $E(Cone) = Cone+Cone$, and $E(c)$ to be the empty category for all other c, and we define τ_1 and τ_2 to force the equations as described above: on most components, the τ's factor through $S_1(c)$, but for one of them, we need to factor through $S_2(c)$.

It then follows that for any $x : 2 \longrightarrow A$, $\phi(x)$ is a limiting cone: given any cone $\sigma : Y \longrightarrow x$, the diagram $\psi(\sigma)$ provides a comparison map; and given any comparison map $f : Y \longrightarrow X$, functoriality of ψ applied to the arrow



in $[Cone, A]$ shows that



commutes, so $f = \gamma_\sigma$.

So an (S, E)-algebra is precisely a small category with assigned binary products. Observe that an (S, E)-algebra map is a functor that sends assigned binary products to assigned binary products.

It is routine to extend the above example to describe small categories with finite products, and a dual gives small categories with finite coproducts; one

can also describe small monoidal categories and symmetric monoidal categories. We can go further than that too, and give all the structures cited in the introduction except for exponentials: our structure here is inherently covariant, whereas exponentials involve some contravariance. In particular, another leading example for us is that there exists algebraic structure (S, E) for which an (S, E)-algebra is precisely a small category together with a monad on it. The construction is not difficult. For instance, for an endofunctor, one just puts $S(c) = 1$ if $c = 1$ and makes it empty otherwise, with no equations. More examples appear in Robinson's paper [17] and in [12].